How to Use pkgDepTools

Seth Falcon

November 20, 2007

1 Introduction

The pkgDepTools package provides tools for computing and analyzing dependency relationships among R packages. With it, you can build a graph-based representation of the dependencies among all packages in a list of CRAN-style package repositories. There are utilities for computing installation order of a given package and, if the RCurl package is available, estimating the download size required to install a given package and its dependencies.

This vignette demonstrates the basic features of the package.

2 Graph Basics

A graph consists of a set of nodes and a set of edges representing relationships between pairs of nodes. The relationships among the nodes of a graph are binary; either there is an edge between a pair of nodes or there is not. To model package dependencies using a graph, let the set of packages be the nodes of the graph with directed edges originating from a given package to each of its dependencies. Figure 1 shows a part of the Bioconductor dependency graph for the Category package. Since circular dependencies are not allowed, the resulting dependency graph will be a directed acyclic graph (DAG).

3 Building a Dependency Graph

```r
> library("pkgDepTools")
> library("Biobase")
```
The `makeDepGraph` function retrieves the meta data for all packages of a specified type (source, win.binary, or mac.binary) from each repository in a list of repository URLs and builds a `graphNEL` instance representing the packages and their dependency relationships.

The function takes four arguments: 1) `repList` a character vector of CRAN-style package repository URLs; 2) `suggests.only` a logical value indicating whether the resulting graph should represent relations from the `Depends` field (FALSE, default) or the `Suggests` field (TRUE); 3) `type` a string indicating the type of packages to search for, the default is `getOption("pkgType")`; 4) `keep.builtin` which will keep packages that come with a standard R install in the dependency graph (the default is FALSE).

Here we use `makeDepGraph` to build dependency graphs of the BioC and CRAN packages. Each dependency graph is a `graphNEL` instance. The out-edges of a given node list its direct dependencies (as shown for package `annotate`). The node attribute “size” gives the size of the package in megabytes when the `dosize` argument is TRUE (this is the default). Obtaining the size of packages requires the RCurl package and can be time consuming for large repositories since a separate HTTP request must be made for each package. In the examples below, we set `dosize=FALSE` to speed the computations.

```r
dl > library("Rgraphviz")

The `makeDepGraph` function retrieves the meta data for all packages of a specified type (source, win.binary, or mac.binary) from each repository in a list of repository URLs and builds a `graphNEL` instance representing the packages and their dependency relationships.

The function takes four arguments: 1) `repList` a character vector of CRAN-style package repository URLs; 2) `suggests.only` a logical value indicating whether the resulting graph should represent relations from the `Depends` field (FALSE, default) or the `Suggests` field (TRUE); 3) `type` a string indicating the type of packages to search for, the default is `getOption("pkgType")`; 4) `keep.builtin` which will keep packages that come with a standard R install in the dependency graph (the default is FALSE).

Here we use `makeDepGraph` to build dependency graphs of the BioC and CRAN packages. Each dependency graph is a `graphNEL` instance. The out-edges of a given node list its direct dependencies (as shown for package `annotate`). The node attribute “size” gives the size of the package in megabytes when the `dosize` argument is TRUE (this is the default). Obtaining the size of packages requires the RCurl package and can be time consuming for large repositories since a separate HTTP request must be made for each package. In the examples below, we set `dosize=FALSE` to speed the computations.

```r
> biocUrl <- biocReposList()$"bioc"
> biocDeps <- makeDepGraph(biocUrl, type="source", dosize=FALSE)
> biocDeps

A graphNEL graph with directed edges
Number of Nodes = 314
Number of Edges = 529

> edges(biocDeps)$"annotate"

$annotate
 [1] "Biobase" "AnnotationDbi" "xtable"

1See help("graphNEL-class")
4 Using the Dependency Graph

The dependencies of a given package can be visualized using the graph generated by `makeDepGraph` and the `Rgraphviz` package. The graph shown in Figure 1 was produced using the code shown below. The `acc` method from the `graph` package returns a vector of all nodes that are accessible from the given node. Here, it has been used to obtain the complete list of `Category`'s dependencies.

```r
> categoryNodes <- c("Category",
+   names(acc(biocDeps, "Category")[[1]])
> categoryGraph <- subGraph(categoryNodes, biocDeps)
> nn <- makeNodeAttrs(categoryGraph, shape="ellipse")
> plot(categoryGraph, nodeAttrs=nn)
```

In R, there is no easy way to preview a given package's dependencies and estimate the amount of data that needs to be downloaded even though the `install.packages` function will search for and install package dependencies if you ask it to by specifying `dependencies=TRUE`. The `getInstallOrder` function provides such a “preview”.

For computing installation order, it is useful to have a single graph representing the relationships among all packages in all available repositories. Below, we create such a graph combining all CRAN and Bioconductor packages.

```r
> allDeps <- makeDepGraph(biocReposList(), type="source",
+   keep.builtin=TRUE, dosize=FALSE)
```

Calling `getInstallOrder` for package `GOstats`, we see a listing of only those packages that need to be installed. Your results will be different based upon your installed packages.

```r
> getInstallOrder("GOstats", allDeps)

$packages
character(0)

$total.size
numeric(0)
```
When `needed.only=FALSE`, the complete dependency list is returned regardless of what packages are currently installed.

```r
> getInstallOrder("GOstats", allDeps, needed.only=FALSE)
```

```r
$packages
[1] "methods" "tools" "stats" "graphics"
[5] "utils" "cluster" "graph" "Biobase"
[9] "DBI" "RSQLite" "AnnotationDbi" "GO.db"
[13] "xtable" "annotate" "RBGL" "splines"
[17] "survival" "genefilter" "Category" "GOstats"

$total.size
[1] NA
```

The edge directions of the dependency graph can be reversed and the resulting graph used to determine the set of packages that make use of (even indirectly) a given package. For example, one might like to know which packages make use of the `methods` package. Here is one way to do that:

```r
> allDepsOnMe <- reverseEdgeDirections(allDeps)
> usesMethods <- dijkstra.sp(allDepsOnMe, start="methods")$distance
> usesMethods <- usesMethods[is.finite(usesMethods)]
> length(usesMethods) - 1 # don't count methods itself
[1] 920
>
```

```r
> table(usesMethods)
```

`usesMethods`

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>352</td>
<td>202</td>
<td>355</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

```r
> toLatex(sessionInfo())
```

- R version 2.6.0 (2007-10-03), x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
• Other packages: Biobase 1.16.1, graph 1.16.1, pkgDepTools 1.4.1, RBGL 1.14.0, RCurl 0.8-3, Rgraphviz 1.16.0

• Loaded via a namespace (and not attached): cluster 1.11.9
Figure 1: The dependency graph for the Category package.